Network Layer

Ping Pong test program

Aug 2, 1989

The Network Layer software supports task-to-task communication across the network. A set of four Pascal-callable interface routines are provided that invoke the Network Layer routines to provide network service to application programs in the VME Local Stations. The PingPong application program is a demonstration of the use of these routines.

Upon program initialization PingPong connects to the network using the two names `PING` and `PONG`. The `PING` "task" accepts operator input to send a request to `PONG`. The `PONG` "task" will respond to any requests that are sent to it. The same application is simultaneously a `PING` task and a `PONG` task. It can be loaded on two different nodes for the purpose. This allows exercising sending requests and receiving replies between two nodes. One can be `PING` and the other will be `PONG`. Or, they can be both simultaneously.

In order to provide for some example "work" to be done by the replying task, the request is interpreted as a request for memory data that should be returned in response to the request. Both the number of bytes and the starting memory address are specified in the last three words of the request message. With the 9-word Acnet header, the size of the request message is therefore 24 bytes. The size of the response message to such a request is 20 plus the number of bytes requested, since the first word of the response (following the Acnet header) is a status word with the value '`OK`' or '`BE`' as an indication of whether a bus error was encountered when accessing the requested memory. The format of the display showing the `PING` activity is as follows:

```
J NET PING PONG   08/01/89 1057
PING   OPEN     0   *CLOSE
      *WRITE    0    0   N<9000>
   0002 0000 7304 0804 504F4E47
   0000 1234 0018 0104 00102008
       READ     0 T=  14 N=9000
   0004 0000 7304 0804 504F4E47
   0003 1234 0118 "OK" 8908 0109
PONG   OPEN     0   *CLOSE
       READ             N=   0


       WRITE            N=   0
```

The upper part of the display shows the `PING` message traffic. Note the fields of the Acnet header. The 0002 signifies a request, the destination node is $73, the destination task name is '`PONG`', the message id is $1234, and the message size is 24 bytes. The source node and the source task id for a request message are filled in by the Network Layer software. The three additional words of the request message following the header specify that `PONG` should reply with 260 bytes of memory data beginning at address $102008. This request was executed 9000 times.

The next section of the display shows the response that was received from the `PONG` task in node $73. The Acnet header is mostly identical in the response message, indicated by the first word value of 0004. The replier had to specify the message size and a status word, but the other fields are left the same as were received in the request. Note that the source task id (actually the destination task id for a reply) has the value $03. That value was also part of the received request and serves to route the reply back to the requesting task, `PING` in this case. The time for the response is shown as 14 counts in units of 0.5 msec; hence, it means 7 msec. This time is measured from just before `PING`'s call to `NetWrite` until just after `PING`'s call to `NetRead` in response to the application's invocation due to the Network event that results from the arrival of the response message. There were 9000 responses received.

A example of the display from the perspective of `PONG` is as follows:

```
J NET PING PONG   08/01/89 1125
PING   OPEN     0   *CLOSE
      *WRITE             N<9000>
   0002 0000 7304 0804 504F4E47
   0000 1234 0018 0104 00102008
       READ       T=     N=   0


PONG   OPEN     0   *CLOSE
       READ     0        N= 535
   0002 0000 0804 7304 504F4E47
   0005 5678 0018 0004 0010200C
       WRITE    0        N= 535
   0004 0000 0804 7304 504F4E47
   0005 5678 0018 "OK" 25310024
```

The lower part of the display shows the `PONG` activity. (The upper part merely shows the example request message last used that is saved across invocations of the page.) Note the fields of the Acnet header sent by `PING` from node $73 this time. The value $5678 was used for the message id in this case and the source task id $05 was filled in by node 73's Network Layer software. Four bytes of data were requested from location $10200C in node 08. The reply message sent by `PONG` is also shown, The status word preceding the four returned memory data bytes is displayed here in Ascii and indicates that there was no bus error accessing memory. A total of 535 requests were replied to.

The two lines of the display which show the `NetOpen` status return allow invoking NetClose to test the status return from that call. One normally won't do this, as the tasks will no longer be connected to the network. Leaving the page also closes the network connection for both PING and PONG.

Overview of Network processing

It may be helpful to understand some of what is going on behind the scenes while PingPong is "doing its thing." PING sends a request message by making a call to NetWrite, specifying the message that it wants to send, including the Acnet header in the first 9 words. It also provides a variable which will be set later to indicate the success of the transmission to the network. NetWrite actually allocates a dynamic memory "message block" to house the some control information used by the network. A pointer to the message is put into the message block, and a pointer to the block is passed to NetQueue, which in turn invokes OUTPQX to place the pointer onto the Output Pointer Queue OUTPQ. NetWrite then calls NetSend (which calls NetXmit) to build the network frame in a circular frame buffer and pass it to the token ring chipset. At this point NetWrite returns to the user application.

Meanwhile, the chipset uses DMA to transfer the frame buffer into its own high speed memory which is able to keep up with the 4 Mbps token ring bandwidth. When it obtains the token from the ring, it transmits the frame. When the frame has circulated around the ring, the transmitting chipset strips it from the ring and emits a new token. At this point the success of the transmission is known, and the chipset generates a transmit interrupt. The network transmit interrupt routine (in module NetInt) records a status code in the user's variable that was passed earlier via the call to NetWrite. The way PING is written, no particular notice of this value is made except at the usual 15 Hz invocations of the application, when the screen is updated with the current value of the variable if it has changed.

On the PONG side of the equation, the arrival of the network message to the chipset results in a DMA transfer into a circular frame receive buffer and an interrupt being delivered to the system. The receive interrupt routine uses the destination SAP to obtain a message queue id from the NETCT table of connected SAPs. In this case, it sends a frame reference message containing a pointer to the frame contents to the ANet message queue. Writing a message into this queue wakes up the ANet Task.

The ANet task analyzes the Acnet header and looks up the destination task name in the NETCT Table. It sends a message reference to the associated message queue (whose name is PONG in this case). Another field in the NETCT entry indicates that it should also send event #4 to the application task to signal it that a network event has occurred. (This was arranged automatically by PONG's NetOpen call.) When the application task is invoked with the

Network event, it calls NetRead (as both PING and PONG, since the Network event may signal the arrival of either a request or a response or both). In this case, a message is received by PONG's call to NetRead, and the request, including the Acnet header, is copied into the user's buffer.

PONG interprets the request and calls its own MemData routine to collect the requested memory data into a reply buffer. The Acnet header is copied to the start of that same buffer, the first word is set to denote a reply message, and the last word of the header is set to indicate the total message size. NetWrite is called to deliver the response. As before, NetWrite allocates a message block and puts a pointer to the user's message into it. Then NetQueue and NetSend are called in turn to "get it out the door." NetSend builds the network frame in the circular frame transmit buffer and hands it off to the chipset. The chipset DMA's the frame into its own fast memory and transmits the frame to the token ring.

Back on the PING side, the response frame is received, and the receive interrupt passes a reference to it to the ANet task, which in turn passes a reference to the message to the application and signals the application task via event #4. The application is invoked and finds a message for PING. The call to NetRead results in the response message being copied from the frame buffer into the user's buffer. The cycle is complete. If the count is not yet exhausted, then NetWrite is called to send the request message again to PONG.

The Network Layer implementation, as is seen from the above discussion, does copy messages in memory. The received data is copied from the circular receive frame buffer into the user's buffer. The transmitted data is copied from the user's buffer into the circular transmit frame buffer.

Statistics have been collected on the performance of the Ping Pong test vehicle. They are listed in the following table:

<u>Cache off</u>

RequesterRequesterRequesterReplier					
#bytesTx to Rx		Rx to Tx	Appl ProgTx delayRep rate		Frames / secKBytes / sec
44.5	2.04.3	1.36.5	154	1	
256	5.02.1	4.32.0	7.2	139	36
512	5.52.1	4.43.5	7.7	130	66
1024	8.42.2	4.65.0	10.6	94	97
2048	14.02.5	5.09.0	16.6	60	123
3072	19.52.8	5.213.0	22.5	44	137
4096	25.53.0	5.416.5	28.5	35	144

<u>Cache on</u>

RequesterRequesterRequesterReplier					
#bytesTx to Rx		Rx to Tx	Appl ProgTx delayRep rate		Frames / secKBytes / sec
42.7	2.03.2	1.34.8	208	1	
256	3.62.0	3.22.0	5.7	175	45
512	5.02.0	3.23.5	7.0	143	73
1024	7.42.2	3.35.0	9.6	104	107
2048	12.02.5	3.69.0	14.5	69	141
3072	17.03.7	3.713.0	19.5	51	158
4096	21.53.0	4.016.5	24.5	41	167

The reference to "Cache on" refers to the 68020 instruction cache in both nodes. The "Tx to Rx" refers to the time from the Tx interrupt of the request message to the Rx interrupt of the response message. The "Rx to Tx" refers to the time from the Rx interrupt of a response to the Tx interrupt of the next request. The application program timing is the timing of the `PING` application. The "Replier Tx Delay" is the time from the replier's `NetXmit` routine handing the frame off to the chipset to the time of the Tx interrupt generated by the chipset when the response message has been completely transmitted around the ring. The "Rep Rate" is the time from one request to the next and should equal the sum of "Tx to Rx" and "Rx to Tx" times. It also corresponds to the measured time to make the request and receive the reply from the application's viewpoint. The last two columns are derived from the measured data. The "Frames / sec" only counts the response frames, not the request frames. The "KBytes / sec" only counts the requested memory data bytes, not the total bytes in the frame.